FIG. 1

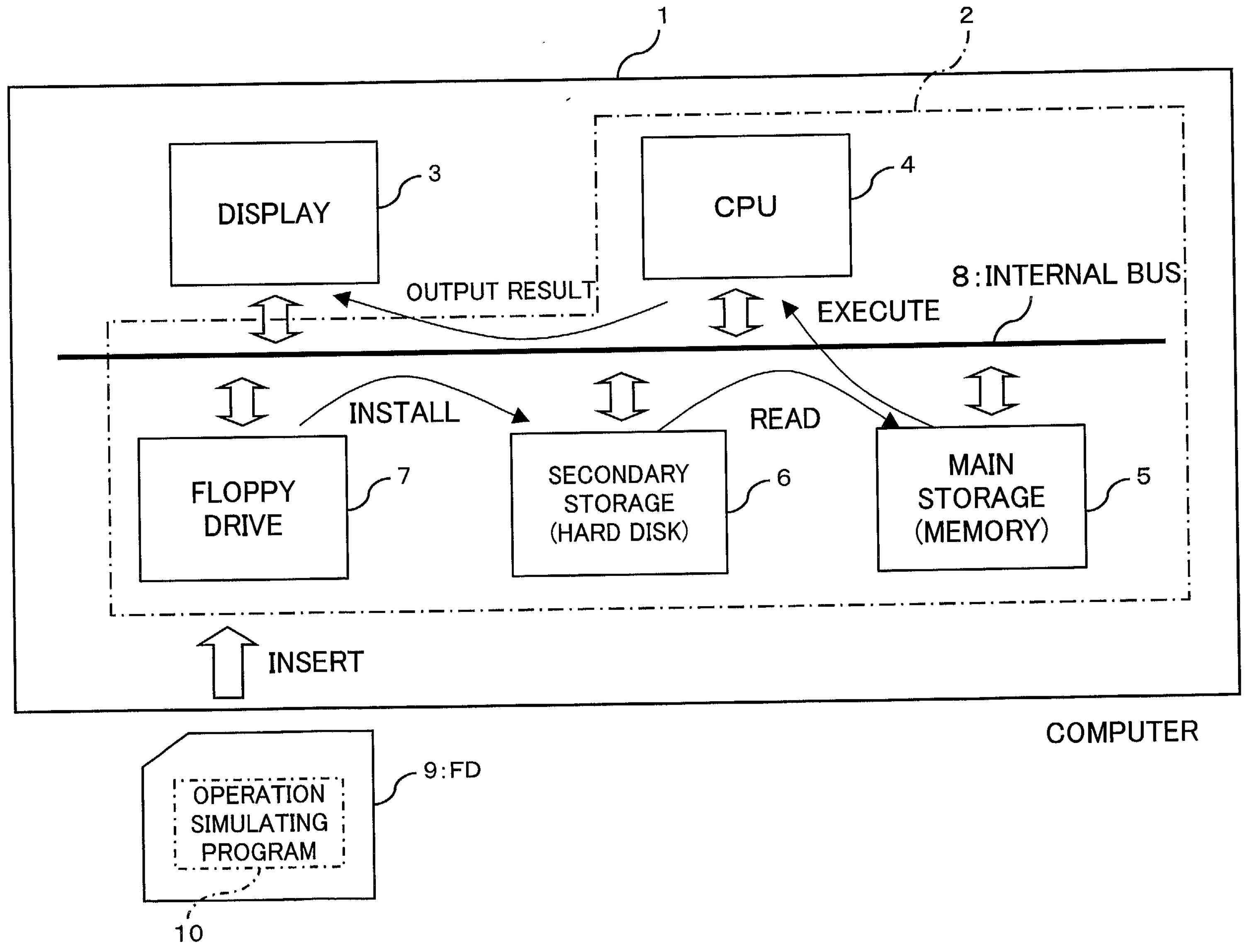

1
2
DISPLAY
3
CPU
4
OUTPUT RESULT
EXECUTE
8:INTERNAL BUS
INSTALL
READ
FLOPPY DRIVE
7
SECONDARY STORAGE (HARD DISK)
6
MAIN STORAGE (MEMORY)
5
INSERT
COMPUTER
OPERATION SIMULATING PROGRAM
9:FD
10

FIG. 2

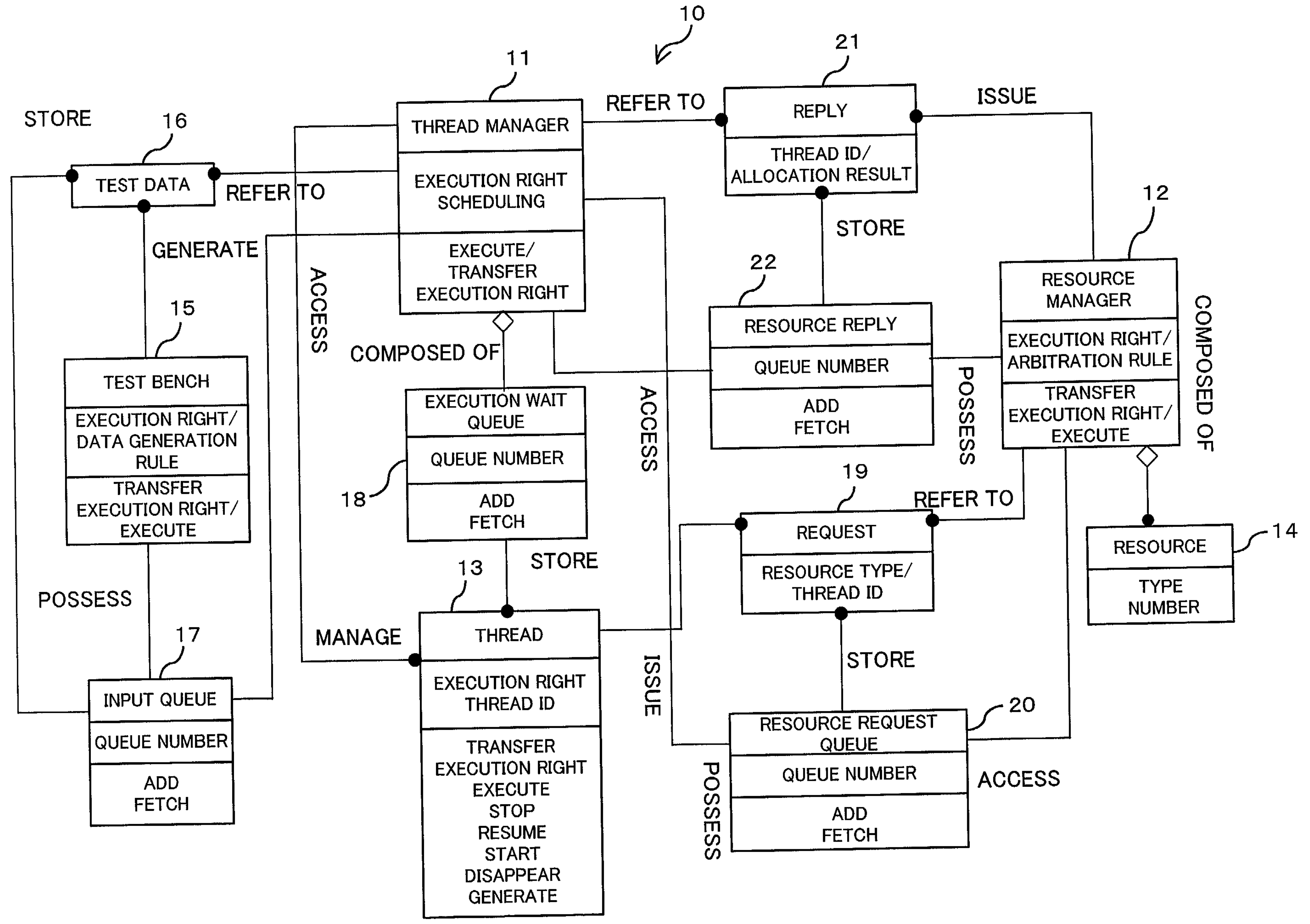

10
11
THREAD MANAGER
EXECUTION RIGHT SCHEDULING
EXECUTE/ TRANSFER EXECUTION RIGHT
REFER TO
21
REPLY
THREAD ID/ ALLOCATION RESULT
ISSUE
12
RESOURCE MANAGER
EXECUTION RIGHT/ ARBITRATION RULE
TRANSFER EXECUTION RIGHT/ EXECUTE
COMPOSED OF
14
RESOURCE
TYPE NUMBER
STORE
22
RESOURCE REPLY
QUEUE NUMBER
ADD FETCH
POSSESS
STORE
16
TEST DATA
REFER TO
GENERATE
15
TEST BENCH
EXECUTION RIGHT/ DATA GENERATION RULE
TRANSFER EXECUTION RIGHT/ EXECUTE
POSSESS
17
INPUT QUEUE
QUEUE NUMBER
ADD FETCH
ACCESS
COMPOSED OF
18
EXECUTION WAIT QUEUE
QUEUE NUMBER
ADD FETCH
STORE
13
THREAD
EXECUTION RIGHT THREAD ID
TRANSFER EXECUTION RIGHT EXECUTE STOP RESUME START DISAPPEAR GENERATE
MANAGE
ACCESS
ISSUE
19
REQUEST
RESOURCE TYPE/ THREAD ID
REFER TO
STORE
20
RESOURCE REQUEST QUEUE
QUEUE NUMBER
ADD FETCH
POSSESS
ACCESS

FIG. 3

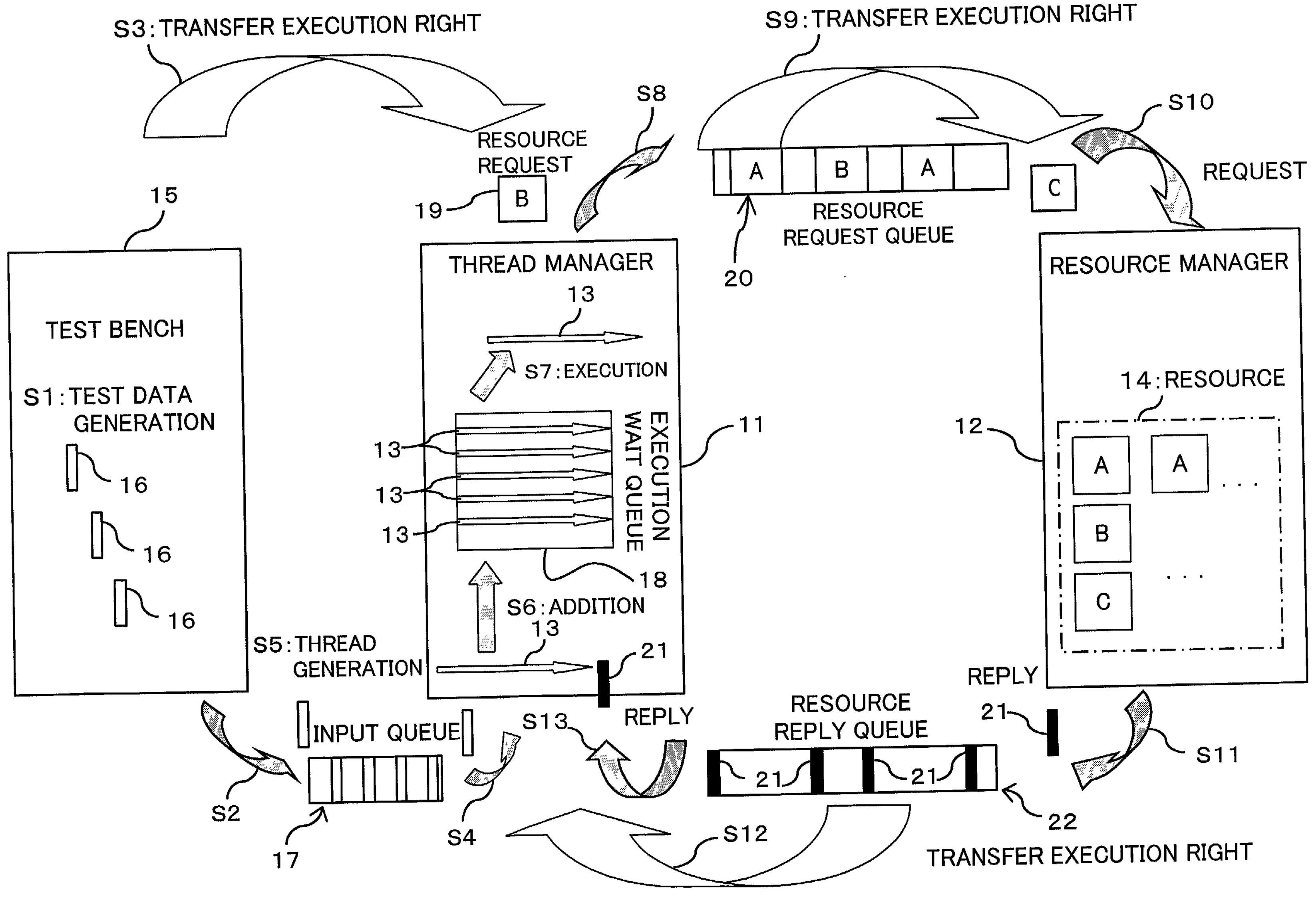
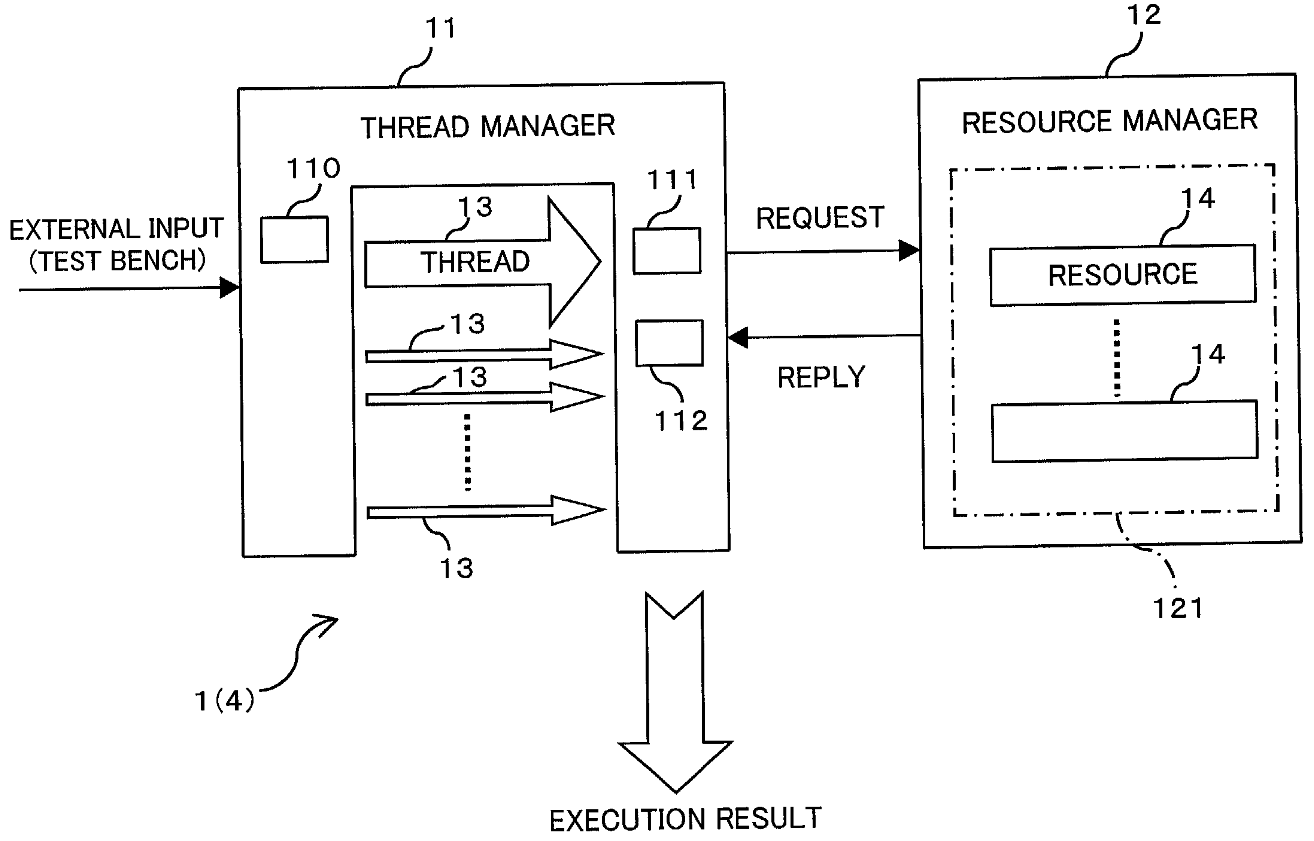

S3: TRANSFER EXECUTION RIGHT
S9: TRANSFER EXECUTION RIGHT
S8
S10
REQUEST
RESOURCE REQUEST
19
B
A
B
A
C
RESOURCE REQUEST QUEUE
20
15
TEST BENCH
S1: TEST DATA GENERATION
16
16
16
THREAD MANAGER
13
S7: EXECUTION
EXECUTION WAIT QUEUE
13
13
13
18
S6: ADDITION
13
21
11
RESOURCE MANAGER
14: RESOURCE
12
A
A
B
C
S5: THREAD GENERATION
REPLY
21
S11
INPUT QUEUE
S13
REPLY
RESOURCE REPLY QUEUE
21
21
22
S2
17
S4
S12
TRANSFER EXECUTION RIGHT

FIG. 4

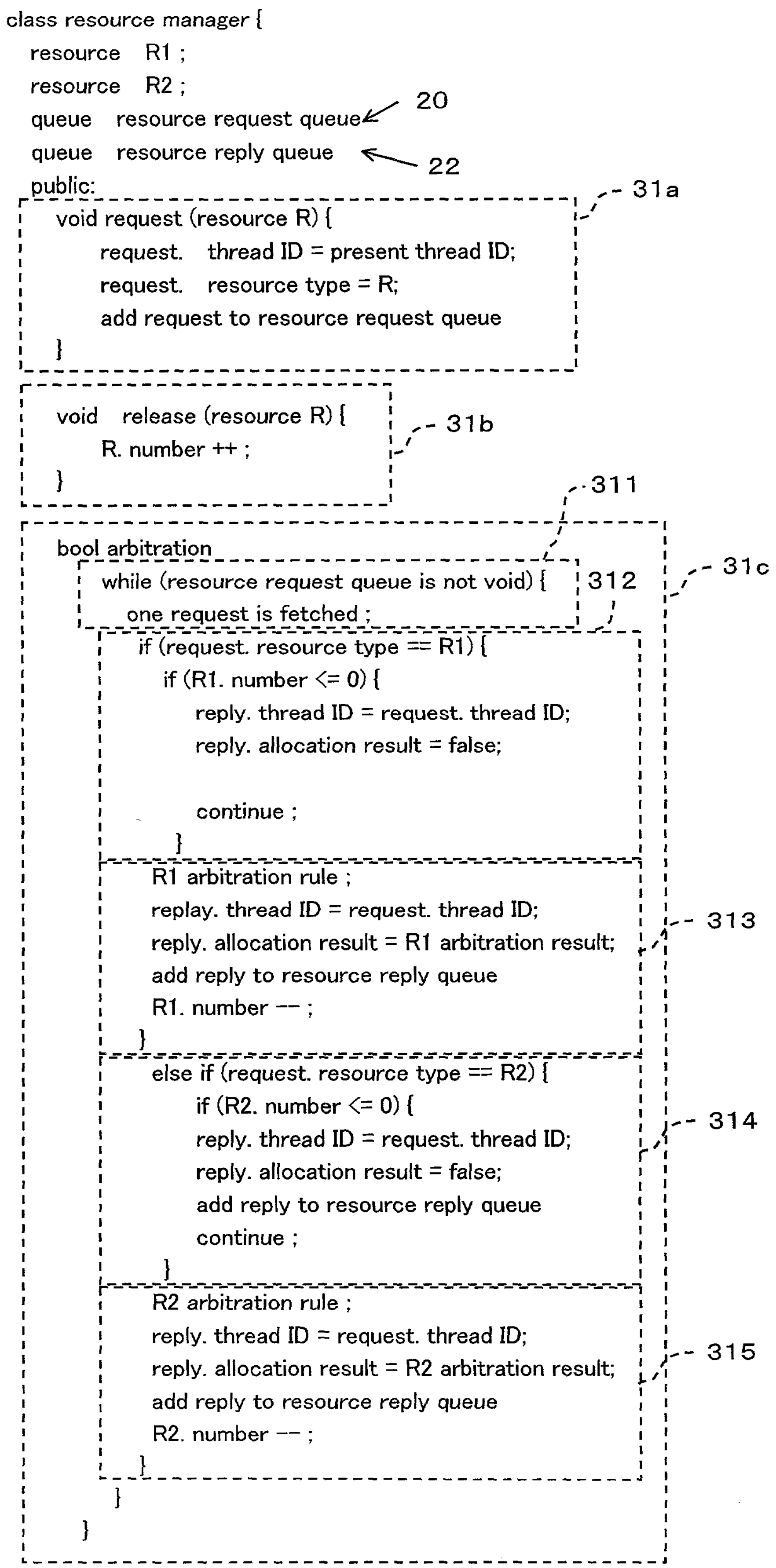


11
THREAD MANAGER
110
EXTERNAL INPUT
(TEST BENCH)
13
THREAD
13
13
13
111
112
REQUEST
REPLY
12
RESOURCE MANAGER
14
RESOURCE
14
121
1(4)
EXECUTION RESULT

FIG. 5

```
class            thread {
   ...
   void execution ( ){
      processing A ( );
      processing B ( );
      processing C ( );
   }
}
```

FIG. 6



```
class resource manager {
  resource   R1 ;
  resource   R2 ;
  queue   resource request queue
  queue   resource reply queue
  public:
    void request (resource R) {
          request.   thread ID = present thread ID;
          request.   resource type = R;
          add request to resource request queue
    }

    void   release (resource R) {
          R. number ++ ;
    }

    bool arbitration
        while (resource request queue is not void) {
           one request is fetched ;
             if (request. resource type == R1) {
               if (R1. number <= 0) {
                   reply. thread ID = request. thread ID;
                   reply. allocation result = false;

                   continue ;
               }
               R1 arbitration rule ;
               replay. thread ID = request. thread ID;
               reply. allocation result = R1 arbitration result;
               add reply to resource reply queue
               R1. number -- ;
             }
               else if (request. resource type == R2) {
                   if (R2. number <= 0) {
                   reply. thread ID = request. thread ID;
                   reply. allocation result = false;
                   add reply to resource reply queue
                   continue ;
                }
               R2 arbitration rule ;
               reply. thread ID = request. thread ID;
               reply. allocation result = R2 arbitration result;
               add reply to resource reply queue
               R2. number -- ;
             }
           }
        }
```

FIG. 7

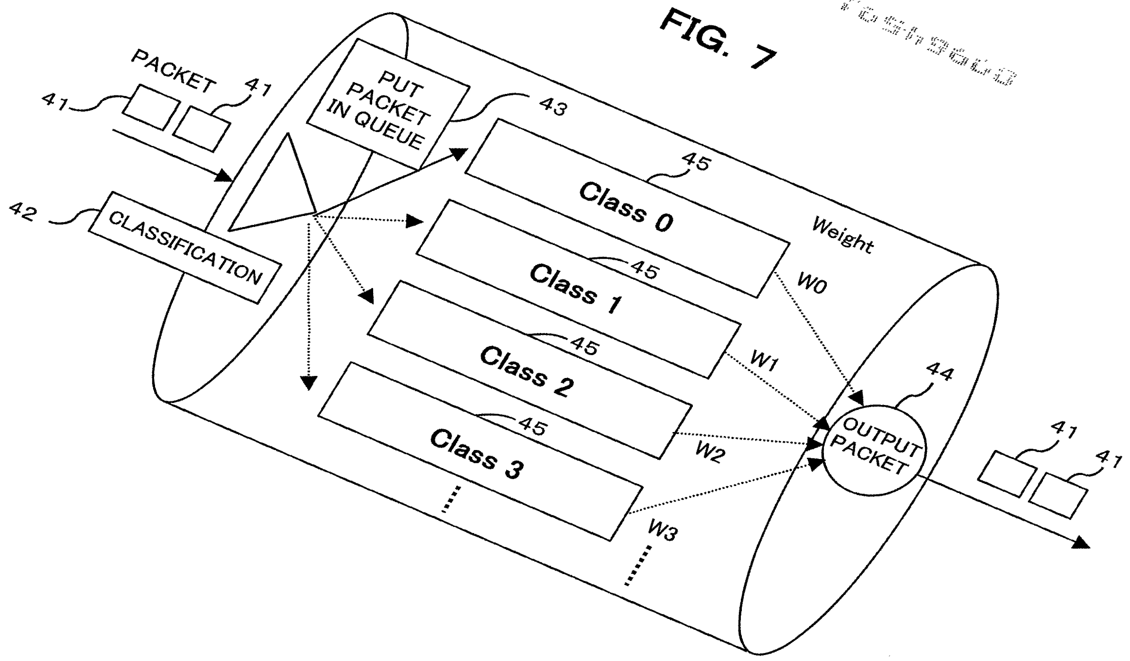

PACKET
41
41
42
CLASSIFICATION
PUT PACKET IN QUEUE
43
45
Class 0
45
Class 1
45
Class 2
45
Class 3
Weight
W0
W1
W2
W3
44
OUTPUT PACKET
41
41

FIG. 8

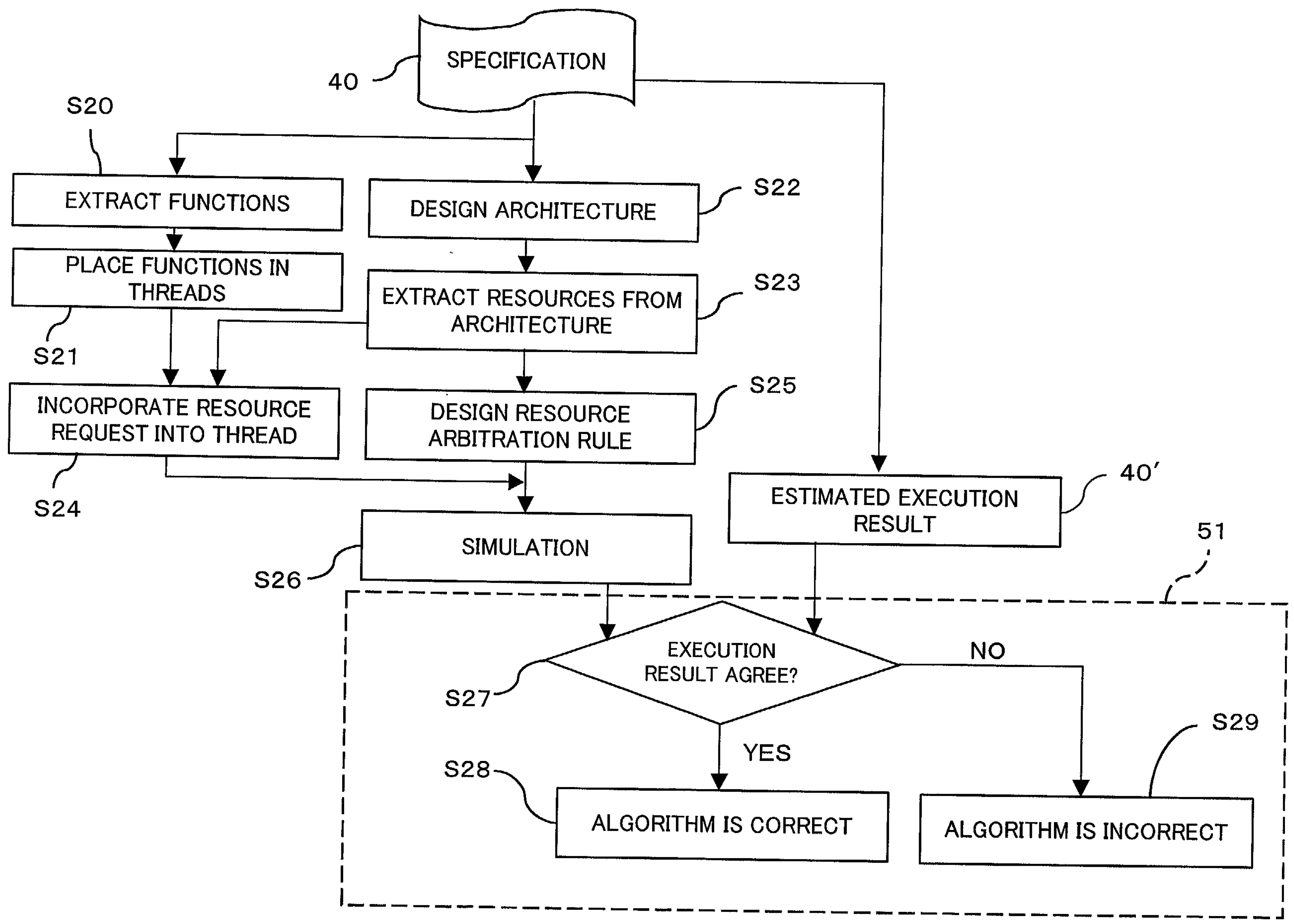

SPECIFICATION
40
S20
EXTRACT FUNCTIONS
PLACE FUNCTIONS IN THREADS
S21
INCORPORATE RESOURCE REQUEST INTO THREAD
S24
DESIGN ARCHITECTURE
S22
EXTRACT RESOURCES FROM ARCHITECTURE
S23
DESIGN RESOURCE ARBITRATION RULE
S25
SIMULATION
S26
ESTIMATED EXECUTION RESULT
40′
51
EXECUTION RESULT AGREE?
S27
NO
YES
S28
ALGORITHM IS CORRECT
S29
ALGORITHM IS INCORRECT

FIG. 9

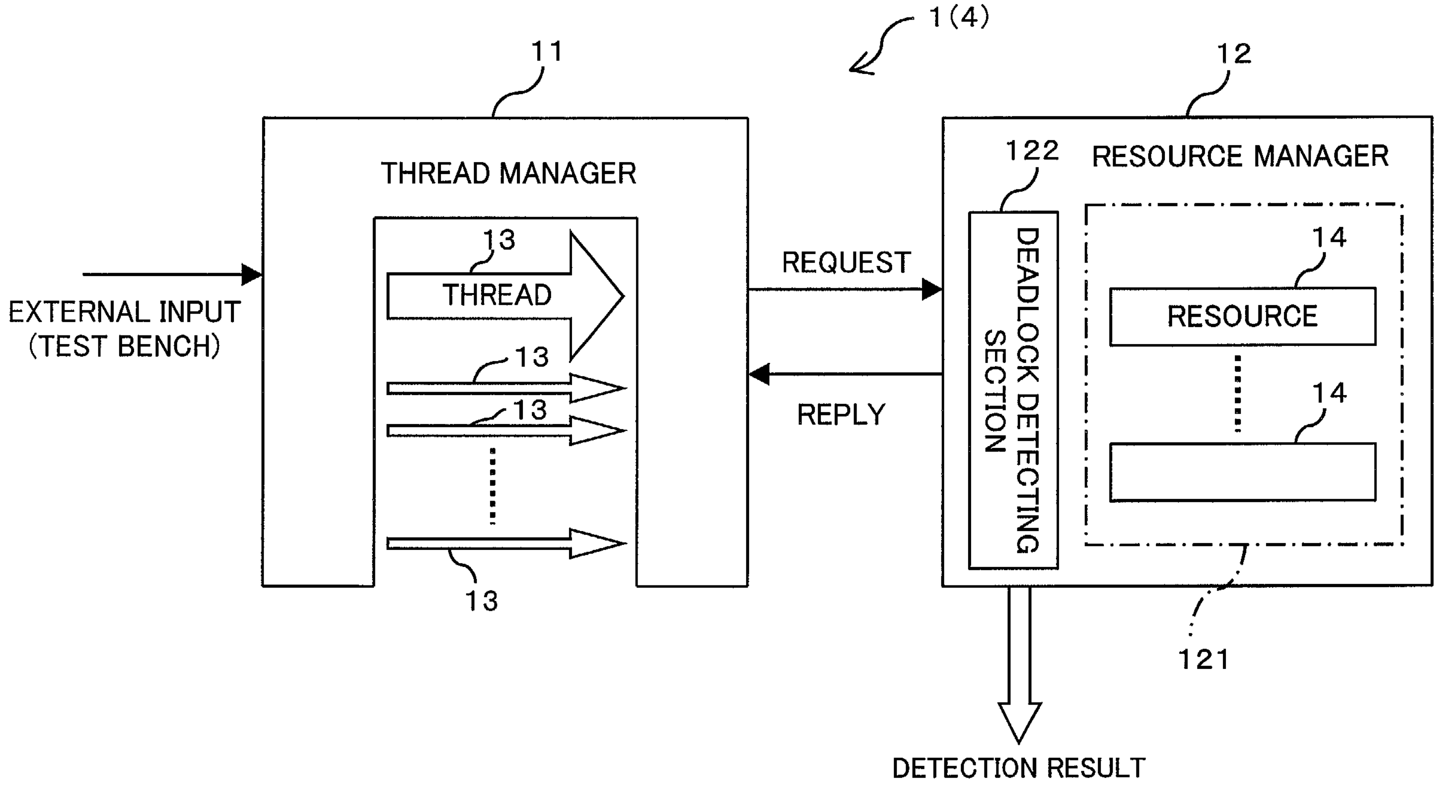
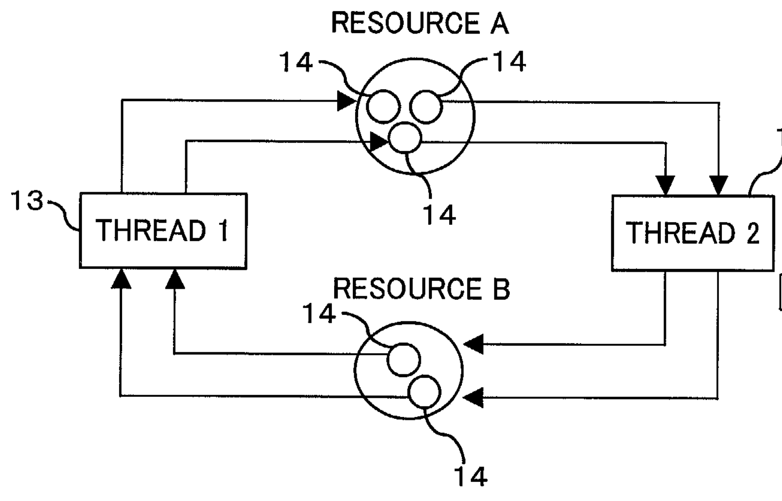

1(4)
11
THREAD MANAGER
13
THREAD
13
13
13
EXTERNAL INPUT
(TEST BENCH)
REQUEST
REPLY
12
RESOURCE MANAGER
122
DEADLOCK DETECTING SECTION
14
RESOURCE
14
121
DETECTION RESULT

FIG. 10

```
class   thread {
   ...
   void   execution ( ){
      resource A. request(2);
      processing;
      ...
resource B. request(2);
resource A. release(2);
      processing;
      ...
      resource A. request(2);
      resource B. release(2);
      processing;
      ...
      resource A. release(2);
   ...
   }
}
```

FIG. 11A



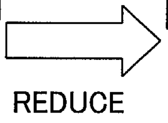


FIG. 11B

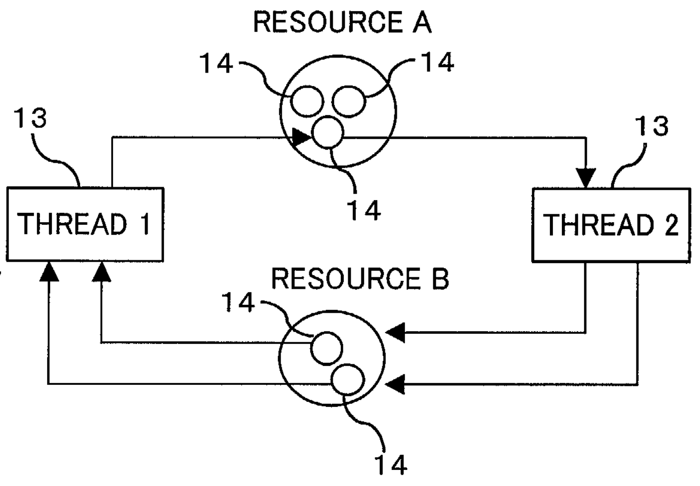


FIG. 12

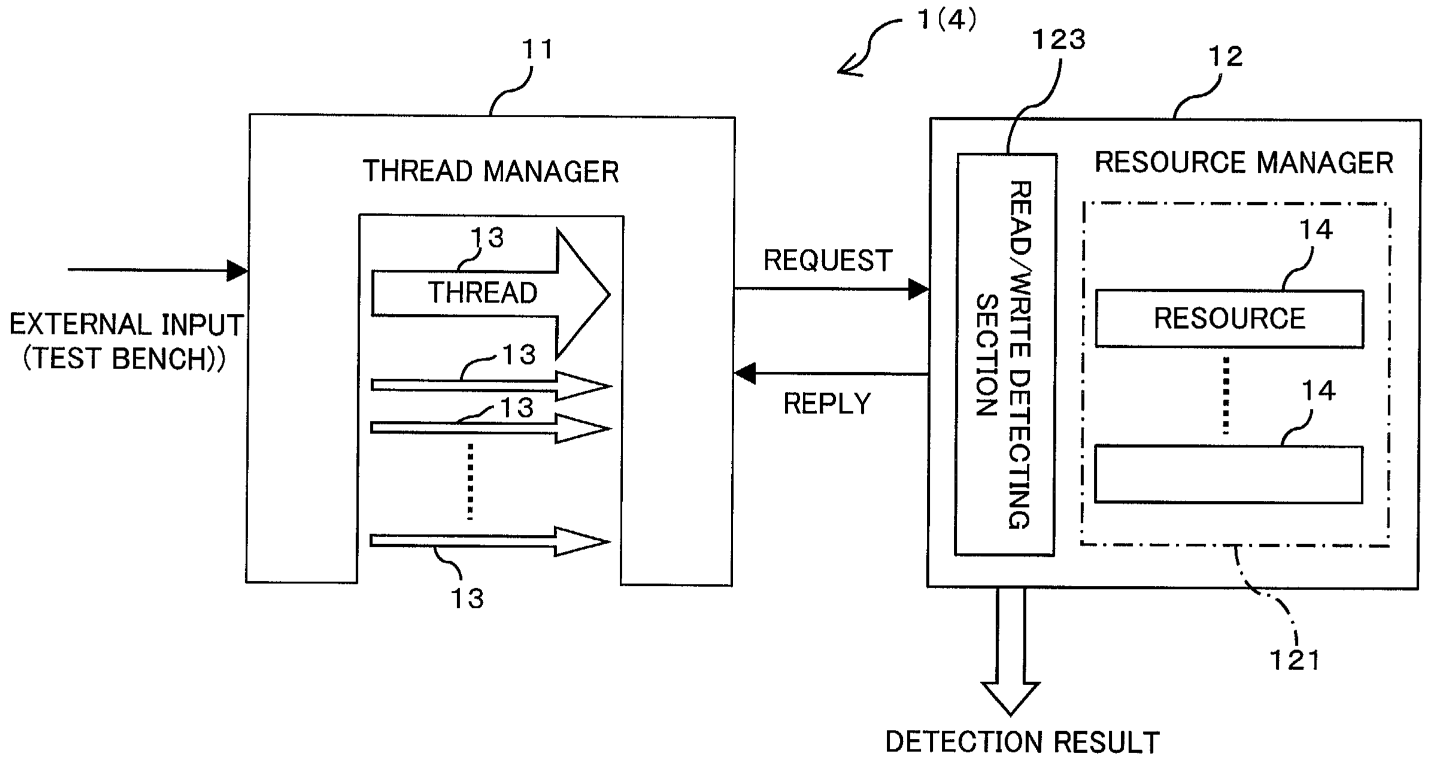

1(4)
11
THREAD MANAGER
13
THREAD
13
13
13
EXTERNAL INPUT (TEST BENCH))
REQUEST
REPLY
123
12
READ/WRITE DETECTING SECTION
RESOURCE MANAGER
14
RESOURCE
14
121
DETECTION RESULT

FIG. 13

```
class   request {
  unsigned int   thread ID;
  int   number of requests;
  int   read/write flag;
}
```

FIG. 14

```
class   resource A : public resource {
int CurrentFlag = 0;
  ...
  void request (int n, bool ReadWriteFlage( ) {
              request. thread ID = present thread ID;
              request.   number of resources = n;
              request.   read/write flag = ReadWriteFlag;
add request to resource request queue
}
void   release (int n){
        ...                                    316
        CurrentFlag = 0;
 }
  ...
bool arbitration( ) {
        ...                                                        317
  while (resource request queue is not void) {
          one request is fetched;
 if (CurrentFlag ! = 0 && request.   read/write flag

     != CurrentFlag){
             error ("there is possibility of occurrence of read/write error!");
           }
       CurrentFlag == request.   read/write flag;
        }
       ...
    }
   ...
}
```

FIG. 15

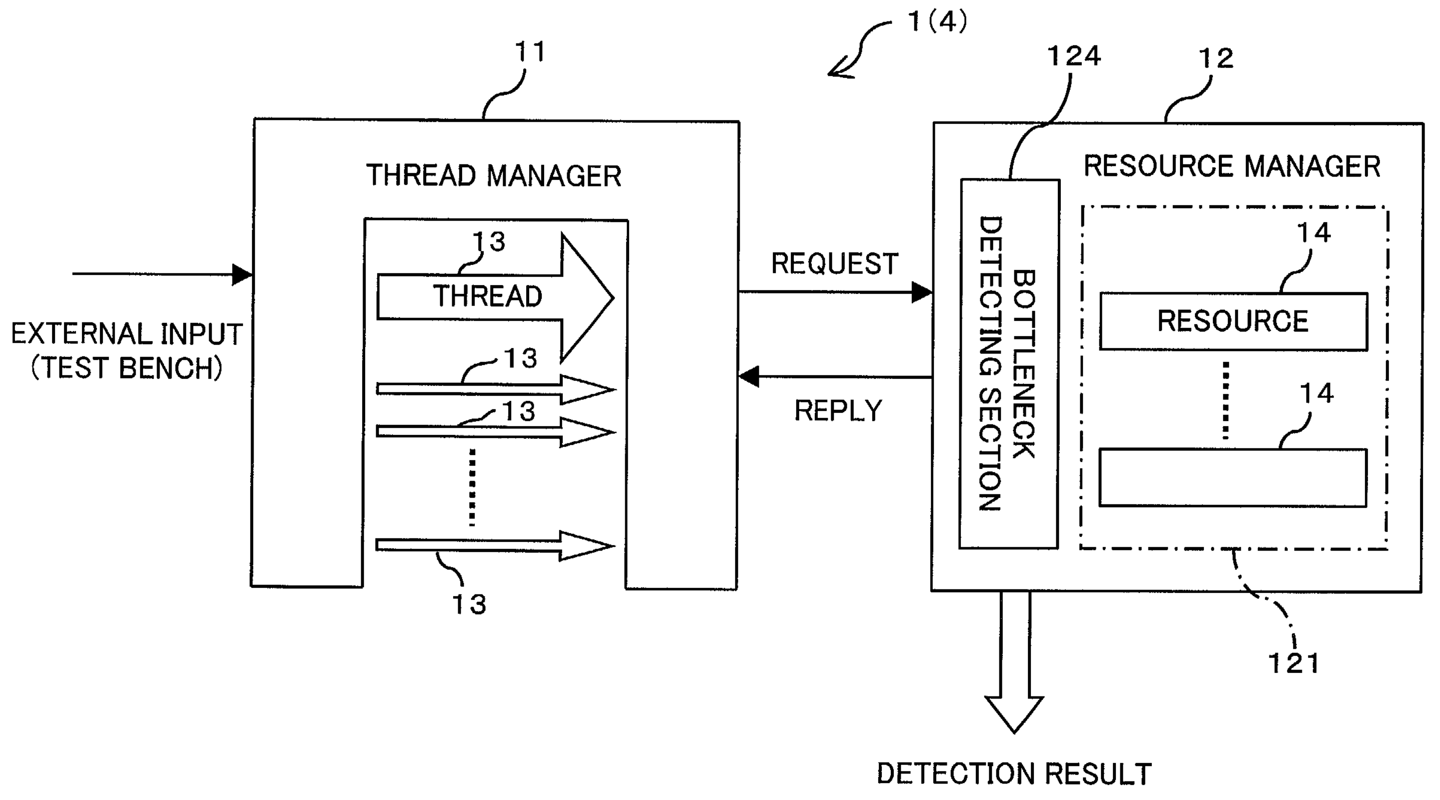

1(4)
11
THREAD MANAGER
13
THREAD
13
13
13
EXTERNAL INPUT
(TEST BENCH)
REQUEST
REPLY
124
12
RESOURCE MANAGER
BOTTLENECK
DETECTING SECTION
14
RESOURCE
14
121
DETECTION RESULT

FIG. 16

```
class    resource {
   int   number of accesses = 0;
   ...
   void   request (int n) {
       number of accesses ++;
   }
   ...
   int    total of request( ) {
       return      number of accesses;
   }
}
```

FIG. 17

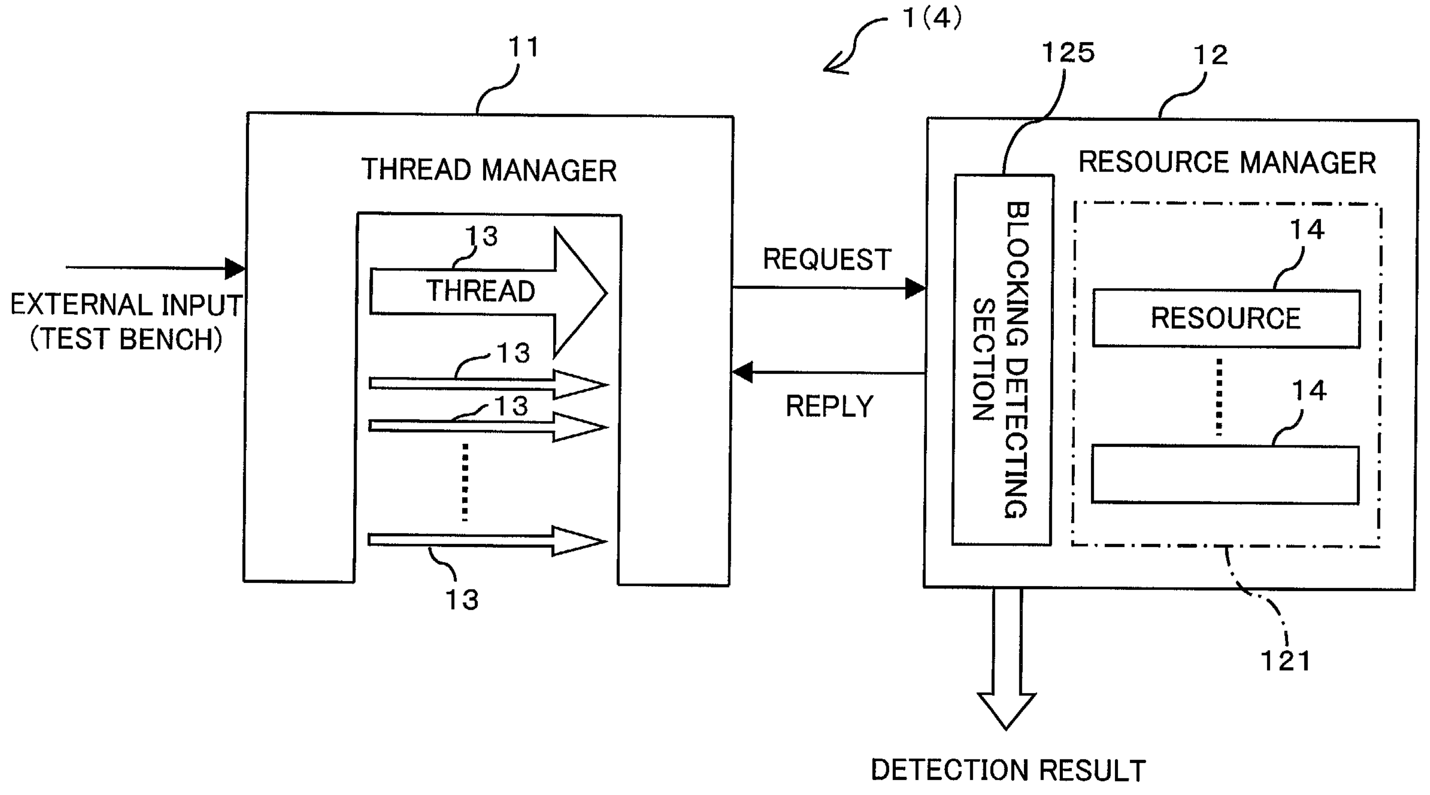

1(4)
11
THREAD MANAGER
13
THREAD
13
13
13
EXTERNAL INPUT
(TEST BENCH)
REQUEST
REPLY
125
12
RESOURCE MANAGER
BLOCKING DETECTING SECTION
14
RESOURCE
14
121
DETECTION RESULT

FIG. 18

```
class    resource A {
      ...
      bool arbitration( ) {
    while (resource request queue is not void) {
        one request is fetched;
if (number < 0) {
   error ("there is a need to block a request for resource A");
}
       }
       ...
 }
```

FIG. 19

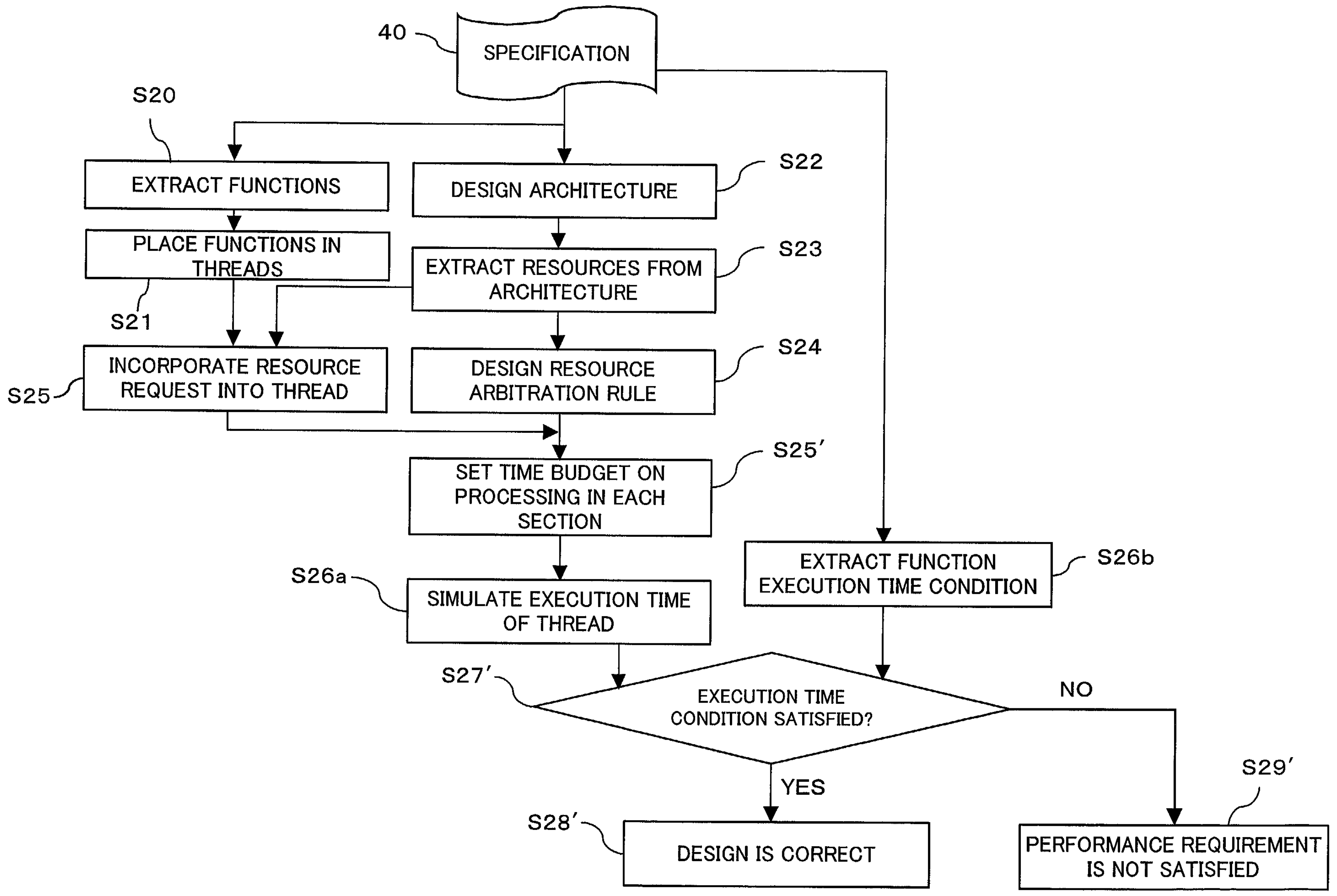
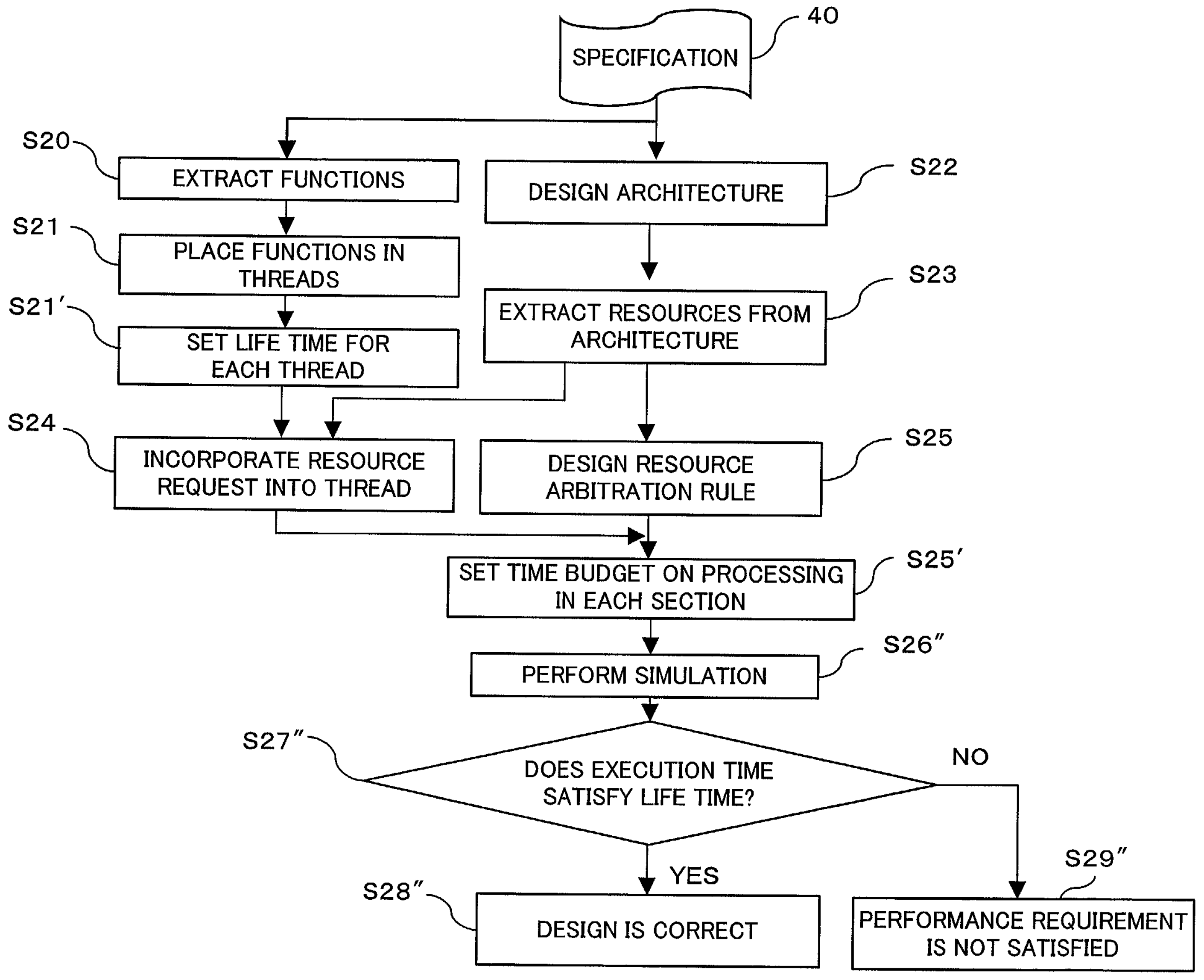

40
SPECIFICATION
S20
EXTRACT FUNCTIONS
PLACE FUNCTIONS IN THREADS
S21
INCORPORATE RESOURCE REQUEST INTO THREAD
S25
DESIGN ARCHITECTURE
S22
EXTRACT RESOURCES FROM ARCHITECTURE
S23
DESIGN RESOURCE ARBITRATION RULE
S24
SET TIME BUDGET ON PROCESSING IN EACH SECTION
S25'
SIMULATE EXECUTION TIME OF THREAD
S26a
EXTRACT FUNCTION EXECUTION TIME CONDITION
S26b
EXECUTION TIME CONDITION SATISFIED?
S27'
YES
NO
DESIGN IS CORRECT
S28'
PERFORMANCE REQUIREMENT IS NOT SATISFIED
S29'

FIG. 20

```
class   thread {
   ...
   void   execution ( ){
      resource A.   request (1);
      processing 1;
      delay (time budget for processing 1);
      resource A.   release (1);
      resource B.   request (1);
      processing 2:
      delay (time budget for processing 2);
      resource B.   release (1);
      ...
   }
}
```

FIG. 21



40
SPECIFICATION
S20
EXTRACT FUNCTIONS
S21
PLACE FUNCTIONS IN THREADS
S21′
SET LIFE TIME FOR EACH THREAD
S24
INCORPORATE RESOURCE REQUEST INTO THREAD
S22
DESIGN ARCHITECTURE
S23
EXTRACT RESOURCES FROM ARCHITECTURE
S25
DESIGN RESOURCE ARBITRATION RULE
S25′
SET TIME BUDGET ON PROCESSING IN EACH SECTION
S26″
PERFORM SIMULATION
S27″
DOES EXECUTION TIME SATISFY LIFE TIME?
YES
NO
S28″
DESIGN IS CORRECT
S29″
PERFORMANCE REQUIREMENT IS NOT SATISFIED

FIG. 22

```
class   thread {
        long   life time;
        long   delay;
        void   execution ( ){
              ...
        }
        bool JugdeLifeTime( ){
              if (life time < delay)
              return false;
              else         return true;
        }
}
```

FIG. 23

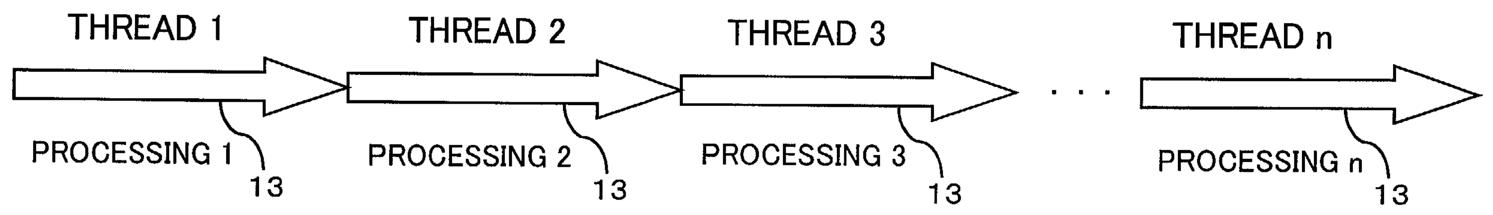

THREAD 1
THREAD 2
THREAD 3
THREAD n
PROCESSING 1
PROCESSING 2
PROCESSING 3
PROCESSING n
13
13
13
13
. . .

FIG. 24

```
class   thread {
   ...
   void   execution( ) {
      thread 1.   generate( );
      thread 1.   wait for completion( );
      thread 2.   generate( );
      thread 2.   wait for completion( );
      thread 3.   generate( );
      thread 3.   wait for completion( );
   }
}

class   thread 1: public thread {
   ...
      void   execution( ) {
      processing 1( );
    }
   ...
}
class   thread 2: public thread {
   ...
   void   execution( ) {
           processing 2( );
        }
     ...
   }

thread 3: public thread {
   ...
   void   execution( ) {
       processing 3( );
   }
   ...
}
```

FIG. 25

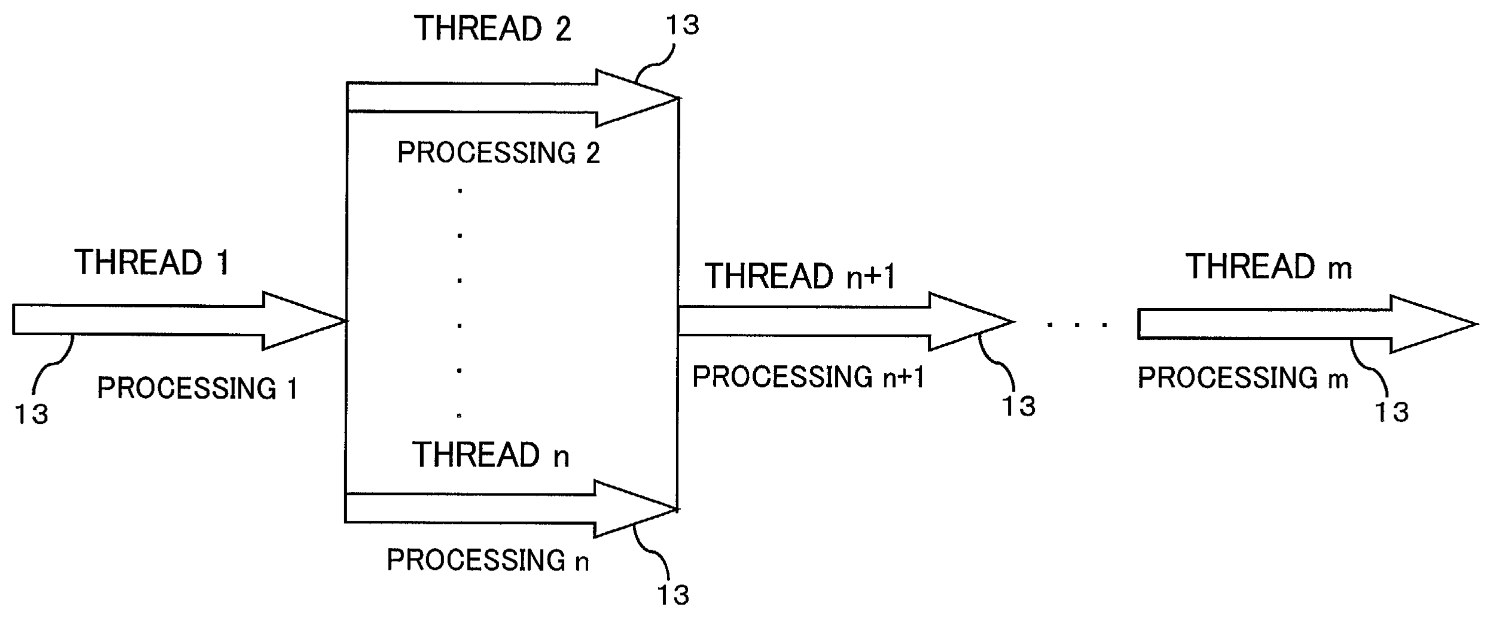
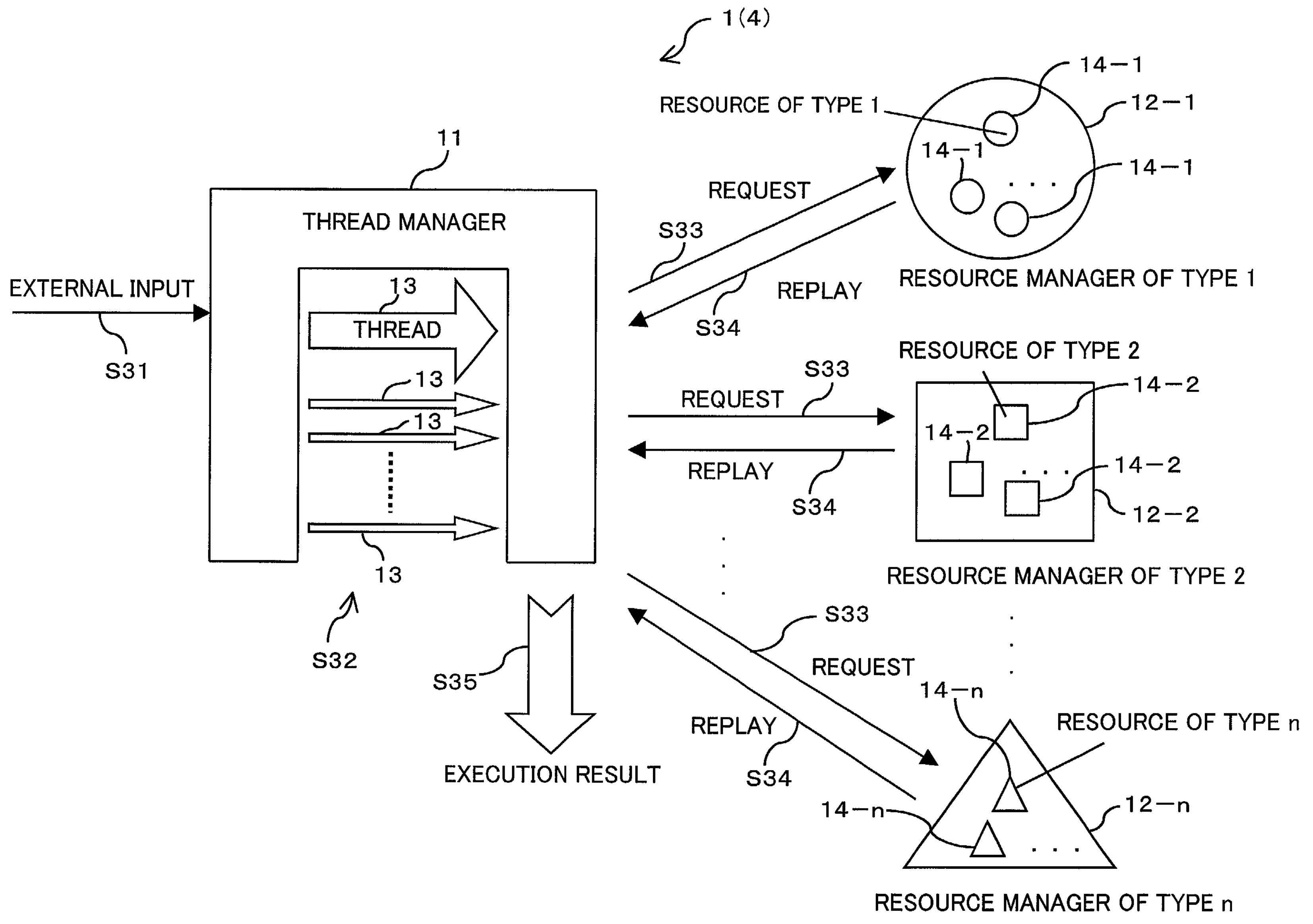

THREAD 1
PROCESSING 1
13
THREAD 2
PROCESSING 2
13
THREAD n
PROCESSING n
13
THREAD n+1
PROCESSING n+1
13
THREAD m
PROCESSING m
13

FIG. 26

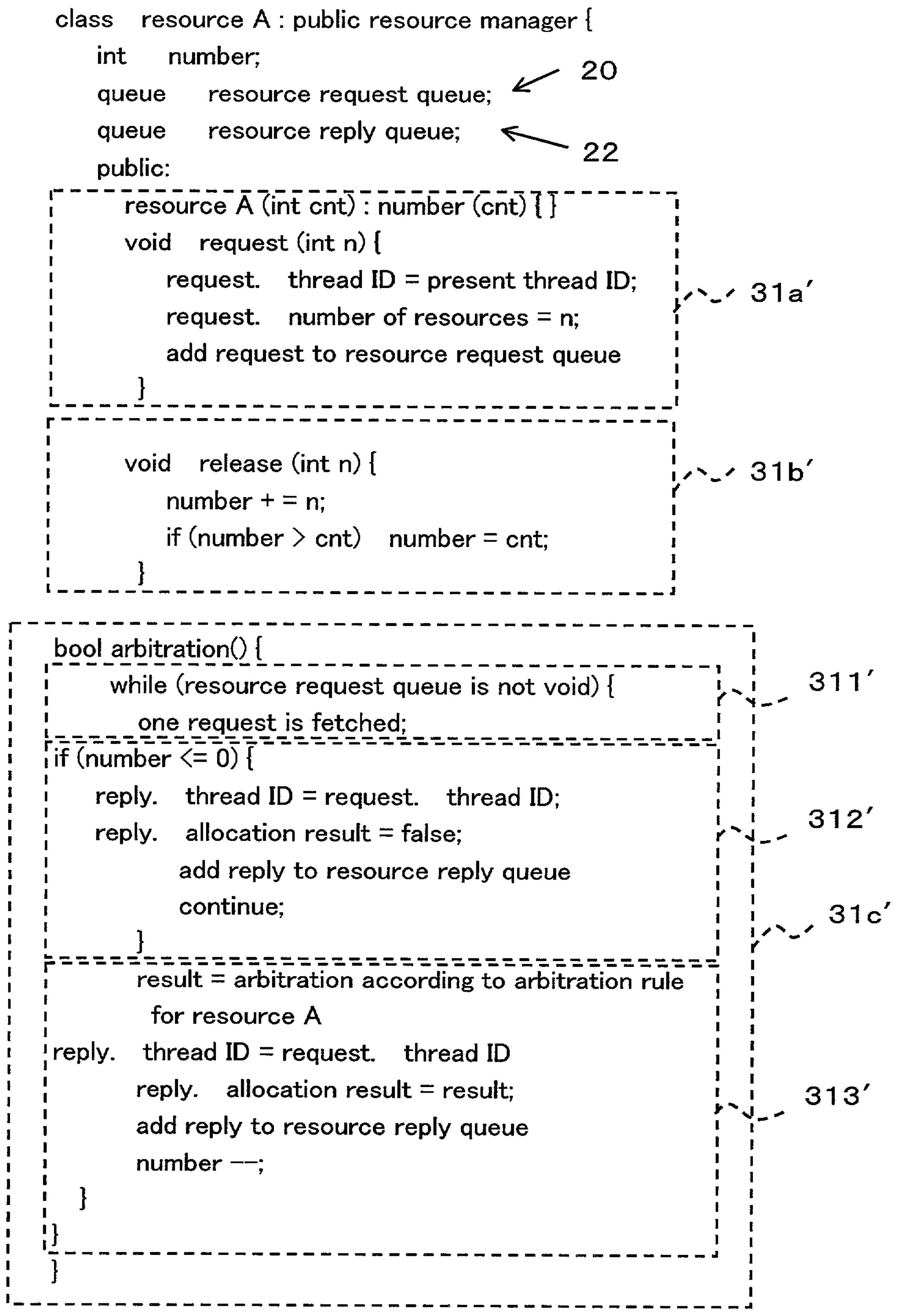
```
class   thread {
   ...
   void   execution( ) {
      thread A.   generate( );
      thread B.    generate( );
      thread A.   wait for completion( );
      thread B.   wait for completion( );
      thread C.   generate( );
      thread C.   wait for completion( );
   }
}
```

FIG. 27



1(4)
11
THREAD MANAGER
EXTERNAL INPUT
S31
13
THREAD
13
13
13
S32
S35
EXECUTION RESULT
RESOURCE OF TYPE 1
14−1
12−1
14−1
14−1
REQUEST
S33
REPLAY
S34
RESOURCE MANAGER OF TYPE 1
RESOURCE OF TYPE 2
S33
REQUEST
14−2
14−2
REPLAY
S34
14−2
12−2
RESOURCE MANAGER OF TYPE 2
S33
REQUEST
14−n
RESOURCE OF TYPE n
REPLAY
S34
14−n
12−n
RESOURCE MANAGER OF TYPE n

FIG. 28



```
class   resource A : public resource manager {
    int     number;
    queue     resource request queue;
    queue     resource reply queue;
    public:
       resource A (int cnt) : number (cnt) { }
       void   request (int n) {
           request.   thread ID = present thread ID;
           request.   number of resources = n;
           add request to resource request queue
       }

       void   release (int n) {
           number + = n;
           if (number > cnt)    number = cnt;
       }

 bool arbitration() {
      while (resource request queue is not void) {
         one request is fetched;
 if (number <= 0) {
     reply.   thread ID = request.   thread ID;
     reply.   allocation result = false;
              add reply to resource reply queue
              continue;
         }
         result = arbitration according to arbitration rule
          for resource A
reply.   thread ID = request.   thread ID
         reply.   allocation result = result;
         add reply to resource reply queue
         number --;
   }
 }
 }
```

09964591.092801

FIG. 29

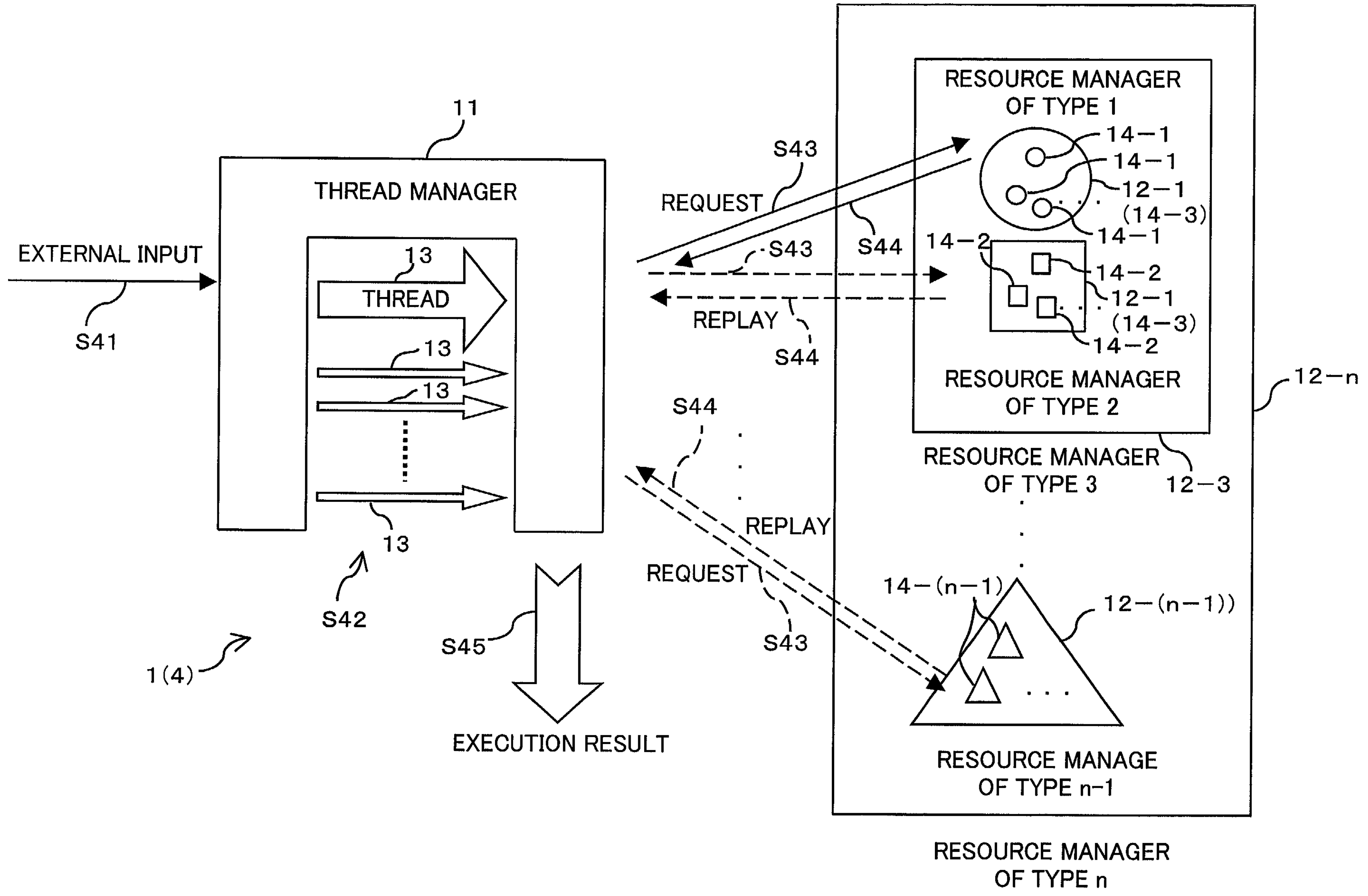


FIG. 30A
RELATED ART

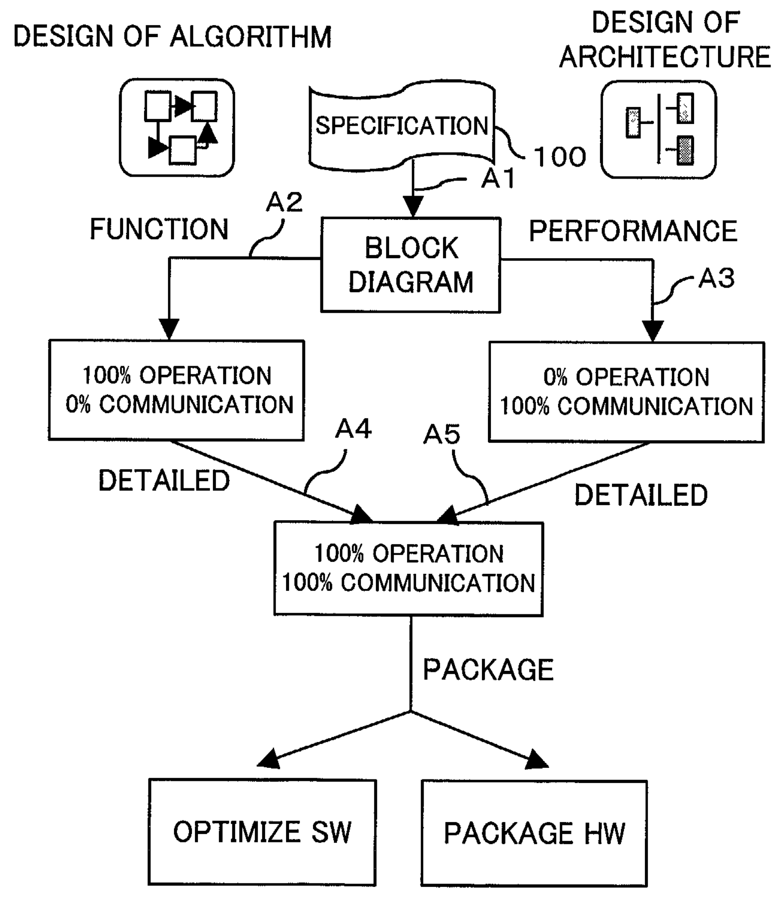


FIG. 30B
RELATED ART

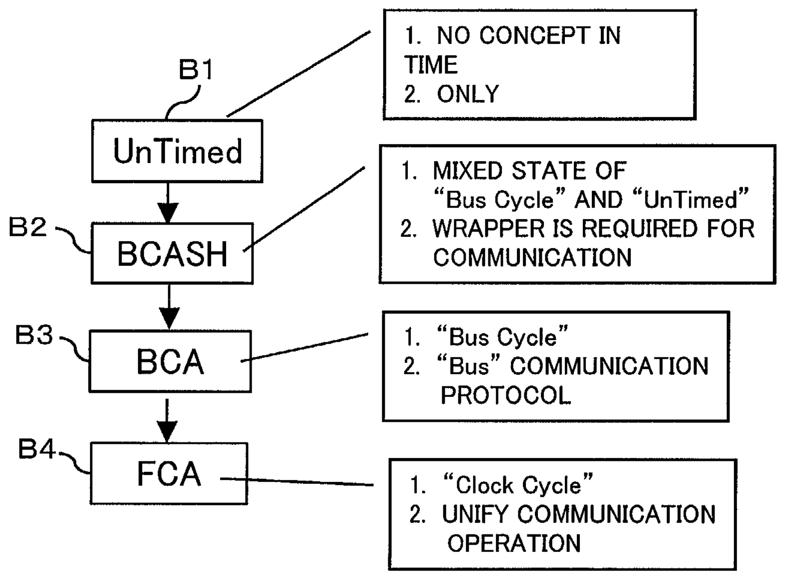
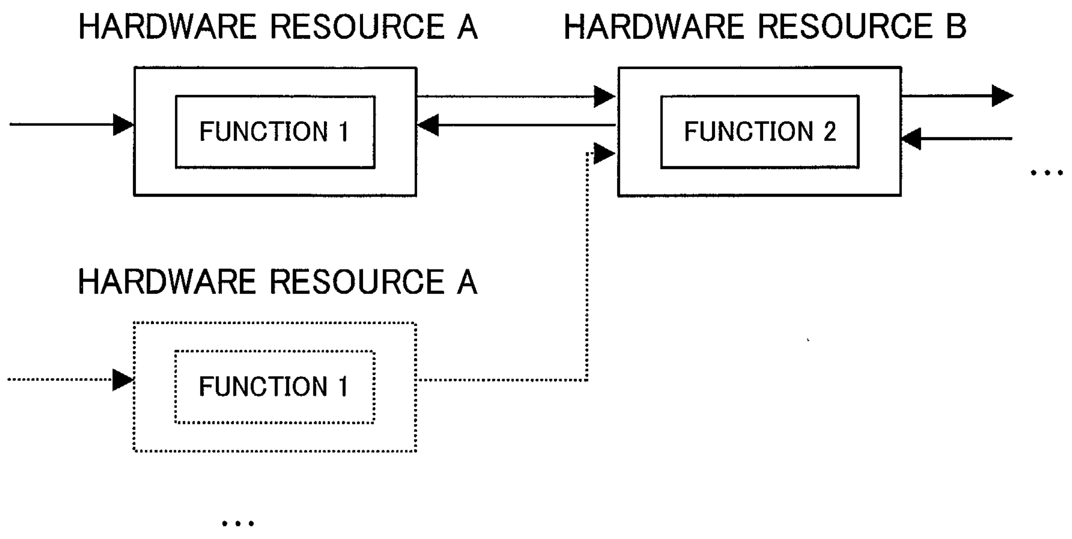

B1
UnTimed
1. NO CONCEPT IN TIME
2. ONLY
B2
BCASH
1. MIXED STATE OF "Bus Cycle" AND "UnTimed"
2. WRAPPER IS REQUIRED FOR COMMUNICATION
B3
BCA
1. "Bus Cycle"
2. "Bus" COMMUNICATION PROTOCOL
B4
FCA
1. "Clock Cycle"
2. UNIFY COMMUNICATION OPERATION

FIG. 31
RELATED ART


HARDWARE RESOURCE A
FUNCTION 1
HARDWARE RESOURCE B
FUNCTION 2
HARDWARE RESOURCE A
FUNCTION 1